Video one: C-style string

```cpp
#include <iostream>
#include <string>
using namespace std;

void printStr(char *str) {
  while(*str) {
    cout << *str;
    ++str;
  }
}

int main() {
  //string s1 = "hello";
  //cout << s1 << endl;

  char s2[6] = "hello";
  printStr(s2); cout << endl;
  cout << s2 << endl;
}
```

```cpp
void printStr(char *str) {
  while(*str) {
    cout << *str;
    ++str;
  }
}

int main() {

  const char *s1 = "frogs";

  char s2[6] = "hello";
  printStr(s2); cout << endl;

  char buffer[1024];
  // put frogs\0 in the buffer
  // put dinosaur\0 in the buffer
}
```

video two：processing c–style strings

Example: strlen() function

```cpp
char str[6] = "hello";
cout << strlen(str) << endl; // Prints 5
```

- Just keep going until we find the **sentinel**.
  - When the current element has value '\0'

```cpp
int strlen(const char *str) {
  const char *ptr = str;
  while (*ptr != '\0') {
    ++ptr;
  }
  return ptr - str;
}
```

Example: count() function

```cpp
char str[6] = "hello";
cout << count(str, 'e') << endl; // Prints 1
cout << count(str, 'l') << endl; // Prints 2
```

```cpp
int count(const char *str, char *c) {
  int count = 0;
  while (*str) {
    if (*str == c) {
      ++count;
    }
    ++str;
  }
  return count;
}
```

Video three: file input and. Output stream

```cpp
int main() {

  string inName = "in.txt";
  string outName = "out.txt";

  cout << "Copying from " << inName << " to " << outName << endl;

  string wordToRemove;
  cout << "What word would you like to remove? ";
  cin >> wordToRemove;

  ifstream fin(inName);
  ofstream fout(outName);
  if ( !fin.is_open() ) {
    cout << "Unable to open " << inName << endl;
    return 1;
  }

  if ( !fout.is_open() ) {
    cout << "Unable to open " << outName << endl;
    return 1;
  }

  string word;
  while (fin >> word) {
    if (word != wordToRemove) { fout << word << " "; }
    else { fout << "*****" << " "; }
  }

  fin.close();
  fout.close();
}
```

Video four: command line argument

```cpp
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main(int argc, char *argv[]) {

  // Usage message shown if the user runs with incorrect command line args
  if (argc != 5) {
    cout << "Usage: redact WORD INFILE OUTFILE NUM_STARS" << endl;
    return 1;
  }

  string inName = argv[2];
  string outName = argv[3];

  cout << "Copying from " << inName << " to " << outName << endl;

  string wordToRemove = argv[1];
  int numStars = atoi(argv[4]); // to double - atof()
  string replacement(numStars, '*'); // e.g. numStars is 3, makes ***

  ifstream fin(inName);
  ofstream fout(outName);
  if ( !fin.is_open() ) {
    cout << "Unable to open " << inName << endl;
    return 1;
  }

  if ( !fout.is_open() ) {
    cout << "Unable to open " << outName << endl;
    return 1;
  }

  string word;
  while (fin >> word) {
    if (word != wordToRemove) { fout << word << " "; }
    else { fout << replacement << " "; }
  }

  fin.close();
  fout.close();
}
```

So that you can use the string functions

argv[0] is the redact

ASCII to integer (change the variable type from char to int)

The reason why we need to use "return 1":
In the terminal, type: echo $?
If the main function runs correct, it will return 0. However, if there are some bugs in main, it will return 1.

When compile the program:
1. g++ redact.cop -o redact
2. ./redact bee in.tex out.tex 10

Video five: the structure of argv

Command Line Arguments

```
$ ./redact bee in.txt out.txt 10
```

- redact is the name of the program to run.
- The other "words" are **arguments** to the redact program.
  - The **shell** (a.k.a. terminal, console, etc.) starts the program and passes arguments.
  - The program gets the arguments. In C++, they are passed as parameters to main.

argv and argc

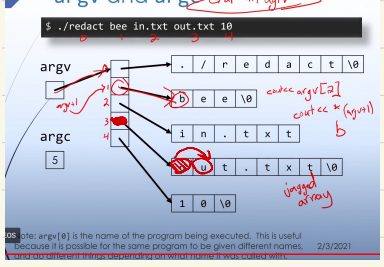
- Two parameters to main:
  - argc – the number of arguments
  - argv – an array of the arguments
- argv is an **array of C-style strings**.

```cpp
int main(int argc, char *argv[]) {

}
```

Compiler turns this into char **argv.

argv and argc

```
$ ./redact bee in.txt out.txt 10
```



Agenda

- Strings
  - C-Style Strings
  - C++ strings
- Command Line Arguments
  - argv and argc
- Stream Input and Output
  - cin and fstreams

Where does the array end?

- What happens if a pointer wanders outside of its array and you use it?
  - Undefined behavior!
  - You end up reading/writing random memory.
  - Program might crash. Or maybe not. Or maybe only sometimes.
- How do we keep pointers in their arrays?
  1. Keep track of the length separately
  2. **Put a sentinel value at the end of the array**

C-Style Strings

- In the old days of the C language, strings were originally represented as just an array of characters.

```cpp
char str1[6] = { 'h', 'e', 'l', 'l', 'o', '\0' };
char str2[6] = "hello";
```

Compiler automatically puts '\0' at the end of string literals.

- There is a **null character** at the end of every string.
  - '\0' in code
  - ASCII value 0
  - Acts as a **sentinel** to say "Whoa, the string stops here!"
- Of course, character arrays turn into pointers as well.

```cpp
char *strPtr = str1;
```

5 C-style Strings are char Arrays

- char values are just numbers underneath

Null character is the sentinel. It has value 0.

ASCII Codes

Symbol	Number
'\0'	0
...	
'e'	101
'f'	102
'g'	103
'h'	104

```cpp
char str[6] = "hello"
```

str

0x804240c0	104
0x804240c1	101
0x804240c2	108
0x804240c3	108
0x804240c4	111
0x804240c5	0

Compiler automatically adds sentinel to string literals.

9/22/2021

6 Be Careful with C-Style Strings

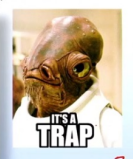
- This code doesn't do what it first appears to. Remember, they turn into pointers.

```cpp
char str1[6] = "hello";
char str2[6] = "hello";
char str3[6] = "apple";
char *ptr = str1;

// Test for equality?
str1 == str2; false

// Copy strings?
str1 = str3;

// Copy through pointer?
ptr = str3;
```

Actually tests if at the same address.

Doesn't compile. Type mismatch.

Makes ptr point to different string.



9/22/2021

Reference: What about C++ strings?

	C-Style Strings	C++ Strings
Library Header	<cstring>	<string>
Declaration	char cstr[]; char *cstr;	string str;
Length	strlen(cstr)	str.length()
Copy value	strcpy(cstr1, cstr2)	str1 = str2
Indexing	cstr[i]	str[i]
Concatenate	strcat(cstr1, cstr2)	str1 += str2
Compare	strcmp(cstr1, cstr2)	str1 == str2

string to C-style string: `const char *cstr = str.c_str();`

C-style string to string: `string str = cstr;`

C-Style Strings and cout

- We saw earlier you can't print out arrays.

```cpp
int array[3] = { 1, 2, 3
cout << array << endl;
```

Turns into an int*. Prints an address, not 1,2,3.

- But you can print out C-style strings.

```cpp
char str[6] = "hello";
cout << str << endl;
```

Turns into a char*. Prints out "hello".

- cout treats ALL char* as C-style strings
  - Starts printing characters until it finds a null character.
  - Don't try to print a char* not pointing into a C-style string!

Example: count() function

```cpp
char str[6] = "hello";
cout << count(str, 'e') << endl; // Prints 1
cout << count(str, 'l') << endl; // Prints 2
```

```cpp
int count(const char *str, char c) {
  int count = 0;
  while (*str) {
    if (*str == c) {
      ++count;
    }
    ++str;
  }
  return count;
}
```

12 Solution: strcpy

```cpp
char word1[5] = "frog";
char word2[7] = "lizard";
strcpy(word2, word1);
cout << word2; // should print "frog"
```

```cpp
void strcpy(char *dst, const char *src) {
  while (*src) {
    *dst = *src;
    ++src;
    ++dst;
  }
  *dst = *src;
}
```

The standard library's strcpy returns the address that was passed in for the first parameter. There are a few uses where this is "convenient". For simplicity, our version returns void.

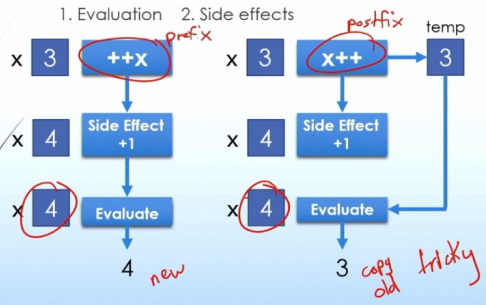
9/22/2021

14

Prefix vs. Postfix Increment

- Parts of an expression

1. Evaluation 2. Side effects



Note: x += 1 is equivalent to ++x

9/22/2021

17

Reference: strcpy (Cute Version)

```
char str1[6] = "hello";
char str2[6] = "apple";
strcpy(str1, str2); // str1 array now holds "apple"
```

```
void strcpy(char *dst, const char *src) {
  while (*dst++ = *src++);
}
```

move
old
old
src
dst

The standard library's strcpy returns the address that was passed in for the first parameter. There are a few uses where this is "convenient". For simplicity, our version returns void.

9/22/2021